Bounded Matrix Low Rank Approximation

Ramakrishnan Kannan, Mariya Ishteva, Haesun Park
School of Computational Science and Engineering
Georgia Institute of Technology
Email:rkannan@gatech.edu and mishteva,hpark@cc.gatech.edu

***Abstract*—Matrix lower rank approximations such as non-negative matrix factorization (NMF) have been successfully used to solve many data mining tasks. In this paper, we propose a new matrix lower rank approximation called Bounded Matrix Low Rank Approximation (BMA) which imposes a lower and an upper bound on every element of a lower rank matrix that best approximates a given matrix with missing elements. This new approximation models many real world problems, such as recommender systems, and performs better than other methods, such as singular value decompositions (SVD) or NMF. We present an efficient algorithm to solve BMA based on coordinate descent method. BMA is different from NMF as it imposes bounds on the approximation itself rather than on each of the low rank factors. We show that our algorithm is scalable for large matrices with missing elements on multi core systems with low memory. We present substantial experimental results illustrating that the proposed method outperforms the state of the art algorithms for recommender systems such as Stochastic Gradient Descent, Alternating Least Squares with regularization, SVD++, Bias-SVD on real world data sets such as Jester, Movielens, Book crossing, Online dating and Netflix.**

***Keywords*-low rank approximation, recommender systems, bound constraints, matrix factorization, block coordinate descent method, scalable algorithm, block**

I. Motivation

In a matrix low rank approximation, given a matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, and a lower rank $k < rank(\mathbf{R})$, we find two matrices $\mathbf{P} \in \mathbb{R}^{n \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times m}$ such that $\mathbf{R}$ is well approximated by $\mathbf{PQ}$, i.e., $\mathbf{R} \approx \mathbf{PQ}$. Low rank approximations vary depending on the constraints imposed on the factors as well as the measure for the difference between $\mathbf{R}$ and $\mathbf{PQ}$. Low rank approximation draws huge interest in the data mining and machine learning community, for it's ability to address many foundational challenges in this area. A few prominent techniques of machine learning that use low rank approximation are principal component analysis, factor analysis, latent semantic analysis, non-negative matrix factorization, etc.

One of the most important low rank approximation is based on singular value decompositions (SVD) [9]. Low rank approximation using SVD has many applications. For example, an image can be compressed by taking the low row rank approximation of its matrix representation using SVD. Similarly, in text mining – latent semantic indexing, is for document retrieval/dimensionality reduction of a term-document matrix using SVD [6]. The other applications include event detection in streaming data, visualization of document corpus etc.

Over the last decade, NMF has emerged as another important low rank approximation technique, where the low-rank factor matrices are constrained to have only non-negative elements. It has received enormous attention and has been successfully applied to a broad range of important problems in areas including text mining, computer vision, community detection in social networks, visualization, bioinformatics etc [18][11].

In this paper, we propose a new type of low rank approximation where the elements of the low rank matrix are bounded – that is, its elements are within a given range which we call as Bounded Matrix Low Rank Approximation(BMA). BMA is different from NMF in that it imposes both upper and lower bounds on its product $\mathbf{PQ}$ rather than each of the low rank factors $\mathbf{P}$ and $\mathbf{Q}$. The goal is to obtain a lower rank approximation $\mathbf{PQ}$ of a given input matrix $\mathbf{R}$, where the elements of $\mathbf{PQ}$ and $\mathbf{R}$ are bounded. There are various real world applications that benefit from this approximation – a well known application being the Recommender System. Currently, the factorization techniques for recommender system, do not use the information that ratings always belong to a range, during the factorization process. Instead, the range is only used to truncate the values of the approximated low rank matrix. In case of recommender system, the input matrix apart from being range bound, also has many missing elements. In general, approximately only 1 or 2% of all matrix elements are known.

We are designing a Bounded Matrix Low Rank Approximation (BMA) that imposes bounds on a low rank matrix that is the best approximate for an input matrix with missing elements.[1] The design considerations are – (1) Simple implementation (2) Scalable to work with any large matrices and (3) Easy parameter tuning with no hyper parameters.

Formally, the BMA problem for an input matrix $\mathbf{R}$ is defined as[2]

[1]Throughout this paper, we might use rows and users, columns and items, reduced rank and hidden latent features, values and ratings interchangeably and appropriately chosen for better understanding of the idea.

[2]Here afterwards, if the inequality is between a vector/matrix and scalar, every element in the vector/matrix should satisfy the inequality against the scalar.

$$\begin{aligned} \min_{\mathbf{P},\mathbf{Q}} \quad & \|\mathbf{M}\cdot*(\mathbf{R}-\mathbf{PQ})\|_F^2 \\ \text{subject to} \quad & \\ & r_{min} \leq \mathbf{PQ} \leq r_{max}. \end{aligned} \tag{1}$$

In the case of input matrix with missing elements, the low rank matrix is approximated only against the known elements of the input matrix. Hence, during error computation the filter matrix $\mathbf{M}$, includes only the corresponding elements of the low rank $\mathbf{PQ}$ for which the values are known. Thus, $\mathbf{M}$ has '1' everywhere for input matrix $\mathbf{R}$ with all known elements. However, in the case of recommender system, the matrix $\mathbf{M}$ has zero for each of the missing elements of $\mathbf{R}$.

Traditionally, regularization is used to control the low rank factors $\mathbf{P}$ and $\mathbf{Q}$ from taking larger values. However, this does not guarantee the value of the product $\mathbf{PQ}$ to be in the given range. We also experimentally show introducing the bounds on the product of $\mathbf{PQ}$ outperforms the low rank approximation algorithms with regularization.

We use Block Coordinate Descent (BCD) framework [2] to solve the Problem (1) as it satisfies the design considerations discussed above. Also, the BCD method provides fine grained control of choosing every element in the low rank factors.

II. NOTATIONS

In this paper, we choose notations consistent with those in the machine learning literature. A lowercase/uppercase such as x or X, is used to denote a scalar; a boldface lowercase letter, such as $\mathbf{x}$, is used to denote a vector; a boldface uppercase letter, such as $\mathbf{X}$, is used to denote a matrix. Indices typically start from 1. When a matrix $\mathbf{X}$ is given, $\mathbf{x}_i$ denotes its i^{th} column, $\mathbf{y}_j^\intercal$ denotes its j^{th} row and x_{ij} or $X(i,j)$ denote its $(i,j)^{th}$ element. For a vector $\mathbf{i}$, $\mathbf{x}(\mathbf{i})$ means vector $\mathbf{i}$ indexes into the elements of vector $\mathbf{x}$. That is, for $\mathbf{x} = [1,4,7,8,10]$ and $\mathbf{i} = [1,3,5]$, $\mathbf{x}(\mathbf{i}) = [1,7,10]$. We have also borrowed certain notations from matrix manipulation scripts such as Matlab/Octave. For example, the $max(\mathbf{x})$ returns the maximal element $x \in \mathbf{x}$ and $max(\mathbf{X})$ returns a vector of maximal elements from each column $\mathbf{x} \in \mathbf{X}$.

For reader's convenience, the Table I is the summary of the notations used in the paper.

III. RELATED WORK

In the literature, two important low rank approximations are NMF and SVD. In this paper, we introduce BMA and its application in recommender systems. Initially, for the input matrix $\mathbf{R} \in \mathbb{R}_+$, we explored the possibility of modelling this problem as a non-negative matrix factorization using BCD. Block coordinate descent has been a commonly used to solve NMF. We briefly survey and explain NMF using BCD framework in Section IV-A.

$\mathbf{R} \in \mathbb{R}^{n\times m}$	Ratings matrix with missing ratings as 0 and ratings bounded within $[r_{min}, r_{max}]$
$\mathbf{M} \in \{0,1\}^{n\times m}$	Indicator matrix with missing ratings as 0
n	Number of users
m	Number of items
k	Reduced rank
$\mathbf{P} \in \mathbb{R}^{n\times k}$	User feature matrix
$\mathbf{Q} \in \mathbb{R}^{k\times m}$	Item feature matrix
$\mathbf{p}_x \in \mathbb{R}^{n\times 1}$	x-th column vector of $\mathbf{P} = (\mathbf{p_1}, \cdots, \mathbf{p_k})$
$\mathbf{q}_x^\intercal \in \mathbb{R}^{1\times m}$	x-th row vector of $\mathbf{Q} = (\mathbf{q_1}, \cdots, \mathbf{q_k})^\intercal$
$r_{max} > 1$	Maximal rating/upper bound
r_{min}	Minimal rating/lower bound
u	A user
i	An item
$\cdot*$	Element wise matrix multiplication
$\cdot/$	Element wise matrix division
$\mathbf{A}(:,i)$	ith column of the matrix $\mathbf{A}$
$\mathbf{A}(i,:)$	ith row of the matrix $\mathbf{A}$
β	Data structure in memory factor
$memsize(v)$	Returns the approximate memory of a variable v =number of elements in v*size of each element*β
μ	Mean of all known ratings in $\mathbf{R}$
$\mathbf{g} \in \mathbb{R}^n$	Bias of all users u
$\mathbf{h} \in \mathbb{R}^m$	Bias of all items i

Table I: Notations

In this section, we explain the important milestones in matrix factorization for recommender systems. Most of these milestones are achieved because of the Netflix competition (http://www.netflixprize.com/) where the winners were awarded 1 million US Dollars as grand prize.

Funk [7] first proposed matrix factorization for recommender system based on SVD and is commonly called as the Stochastic Gradient Descent (SGD). Paterek [23] improved SGD as a combination of the baseline estimate and matrix factorization. Koren, the member of winning team of Netflix prize improved the results with his remarkable contributions in this area. Koren [15] proposed a baseline estimate based on mean rating, user–movie bias, combined with matrix factorization and called it Bias-SVD. In SVD++ [15], he extends this Bias-SVD with implicit ratings and considered only the relevant neighbourhood items during matrix factorization. The Netflix dataset also provided the time of rating. However most of the techniques did not include time in their model. Koren [16] proposes time-svd++, where he extended his previous SVD++ model to include the time. So far, all the techniques discussed here, in matrix factorization for recommender systems are based on SVD and used gradient descent to solve the problem. Alternatively, Zhou et.al., [26] used alternating least squares with regularization (ALSWR). Apart from these directions, there had been other approaches such as Bayesian tensor factorization [25], Bayesian probabilistic modelling [24], graphical modelling of the recommender system problem [21] and weighted low-rank approximation with zero weights for the missing values [22]. A detailed survey and overview of matrix factorization for recommender systems is discussed in [17].

A. Our Contributions

Given the above background, we highlight the contributions of this paper. In this paper, we propose a novel matrix lower rank approximation called Bounded Matrix Low Rank Approximation (BMA) which imposes a lower and an upper bound on every element of a lower rank matrix that best approximates a given matrix with missing elements. We solve the BMA using block coordinate descent method.[3] We also ensure it works seamlessly for recommender system. From this perspective, this is the first work that uses the block coordinate descent method and experiment BMA for recommender systems. We present the details of the algorithm with supporting technical details and a scalable version of the naive algorithm. Also, we test our algorithm on recommender system datasets and compare against state of the art algorithms SGD, SVD++, ALSWR, Bias-SVD.

IV. Foundations

In the case of low rank approximation using NMF, the low rank factor matrices are constrained to have only nonnegative elements. However, in the case of BMA, we constrain the elements of their product with an upper and lower bound rather than each of the two low rank factor matrices. In this section, we explain the BCD framework for NMF and subsequently explain using BCD to solve BMA.

A. NMF and Block Coordinate Descent

In this section, we will see relevant foundation for using BCD framework to solve NMF.

Consider a constrained non-linear optimization problem as follows:

$$\min f(x) \ \text{ subject to } \ x \in \mathcal{X}, \tag{2}$$

where $\mathcal{X}$ is a closed convex subset of $\mathbb{R}^n$. An important assumption to be exploited in the BCD method is that set $\mathcal{X}$ is represented by a Cartesian product:

$$\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m, \tag{3}$$

where $\mathcal{X}_j$, $j = 1, \cdots, m$, is a closed convex subset of $\mathbb{R}^{N_j}$ satisfying $n = \sum_{j=1}^{m} N_j$. Accordingly, vector $\mathbf{x}$ is partitioned as $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_m)$ so that $\mathbf{x}_j \in \mathcal{X}_j$ for $j = 1, \cdots, m$. The BCD method solves for $\mathbf{x}_j$ fixing all other subvectors of $\mathbf{x}$ in a cyclic manner. That is, if $\mathbf{x}^{(i)} = (\mathbf{x}_1^{(i)}, \cdots, \mathbf{x}_m^{(i)})$ is given as the current iterate at the i^{th} step, the algorithm generates the next iterate $\mathbf{x}^{(i+1)} = (\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_m^{(i+1)})$ block by block, according to the solution of the following subproblem:

$$\mathbf{x}_j^{(i+1)} \leftarrow \underset{\xi \in \mathcal{X}_j}{\text{argmin}} f(\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_{j-1}^{(i+1)}, \xi, \mathbf{x}_{j+1}^{(i)}, \cdots, \mathbf{x}_m^{(i)}). \tag{4}$$

Also known as a *non-linear Gauss-Seidel* method [2], this algorithm updates one block each time, always using the most recently updated values of other blocks $\mathbf{x}_{\tilde{j}}, \tilde{j} \neq j$. This is important since it ensures that after each update the objective function value does not increase. For a sequence $\{\mathbf{x}^{(i)}\}$ where each $\mathbf{x}^{(i)}$ is generated by the BCD method, the following property holds.

Theorem 1: Suppose f is continuously differentiable in $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$, where $\mathcal{X}_j$, $j = 1, \cdots, m$, are closed convex sets. Furthermore, suppose that for all j and i, the minimum of

$$\min_{\xi \in \mathcal{X}_j} f(\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_{j-1}^{(i+1)}, \xi, \mathbf{x}_{j+1}^{(i)}, \cdots, \mathbf{x}_m^{(i)})$$

is uniquely attained. Let $\{\mathbf{x}^{(i)}\}$ be the sequence generated by the block coordinate descent method as in Eq. (4). Then, every limit point of $\{\mathbf{x}^{(i)}\}$ is a stationary point. The uniqueness of the minimum is not required when m is two.

The proof of this theorem for an arbitrary number of blocks is shown in Bertsekas [2]. For a non-convex optimization problem, most algorithms only guarantee the stationarity of a limit point [19].

When applying the BCD method to a constrained nonlinear programming problem, it is critical to wisely choose a partition of $\mathcal{X}$, whose Cartesian product constitutes $\mathcal{X}$. An important criterion is whether the subproblems in Eq. (4) are efficiently solvable: For example, if the solutions of subproblems appear in a closed form, each update can be computed fast. In addition, it is worth checking how the solutions of subproblems depend on each other. The BCD method requires that the most recent values need to be used for each subproblem in Eq (4). When the solutions of subproblems depend on each other, they have to be computed sequentially to make use of the most recent values; if solutions for some blocks are independent from each other, however, simultaneous computation of them would be possible. We discuss how different choices of partitions lead to different NMF algorithms. Three cases of partitions are discussed below.

1) BCD with Two Matrix Blocks - ANLS Method: For convenience, we assume all the elements of the input matrix are known and hence ignoring the $\mathbf{M}$ from the discussion. The most natural partitioning of the variables is the two biggest blocks $\mathbf{P}$ and $\mathbf{Q}$ representing the entire matrix. In this case, following the BCD method in Eq. (4), we take turns solving

$$\mathbf{P} \leftarrow \underset{\mathbf{P} \geq 0}{\arg\min} f(\mathbf{P}, \mathbf{Q}) \ \text{ and } \ \mathbf{Q} \leftarrow \underset{\mathbf{Q} \geq 0}{\arg\min} f(\mathbf{P}, \mathbf{Q}). \tag{5}$$

Since the subproblems are the nonnegativity constrained least squares (NLS) problems, the two-block BCD method has been called the alternating nonnegative least square (ANLS) framework [19], [12], [14].

[3]There could be other ways to solve this problem.

2) BCD with 2k Vector Blocks - HALS/RRI Method: Let us now partition the unknowns into 2k blocks in which each block is a column of $\mathbf{P}$ or a row of $\mathbf{Q}$. In this case, it is easier to consider the objective function in the following form:

$$f(\mathbf{p}_1, \cdots, \mathbf{p}_k, \mathbf{q}_1^\intercal, \cdots, \mathbf{q}_k^\intercal) = \|\mathbf{R} - \sum_{j=1}^{k} \mathbf{p}_j \mathbf{q}_j^T\|_F^2, \quad (6)$$

where $\mathbf{P} = [\mathbf{p}_1, \cdots \mathbf{p}_k] \in \mathbb{R}_+^{n \times k}$ and $\mathbf{Q} = [\mathbf{q}_1, \cdots, \mathbf{q}_k]^\intercal \in \mathbb{R}_+^{k \times m}$. The form in Eq. (6) represents that $\mathbf{R}$ is approximated by the sum of k rank-one matrices.

Following the BCD scheme, we can minimize f by iteratively solving

$$\mathbf{p}_i \leftarrow \arg\min_{\mathbf{p}_i \geq 0} f(\mathbf{p}_1, \cdots, \mathbf{p}_k, \mathbf{q}_1^\intercal, \cdots, \mathbf{q}_k^\intercal)$$

for $i = 1, \cdots, k$, and

$$\mathbf{q}_i^\intercal \leftarrow \arg\min_{\mathbf{q}_i^\intercal \geq 0} f(\mathbf{p}_1, \cdots, \mathbf{p}_k, \mathbf{q}_1^\intercal, \cdots, \mathbf{q}_k^\intercal)$$

for $i = 1, \cdots, k$.

The 2K-block BCD algorithm has been studied as Hierarchical Alternating Least Squares (HALS) proposed by Cichocki et.al [5], [4] and independently by Ho [10] as rank-one residue iteration (RRI).

3) BCD with k(n + m) Scalar Blocks: We can also partition the variables with the smallest $k(m+n)$ element blocks of scalars, where every element of $\mathbf{P}$ and $\mathbf{Q}$ is considered as a block in the context of Theorem 1. To this end, it helps to write the objective function as a quadratic function of scalar p_{ij} or q_{ij} assuming all other elements in $\mathbf{P}$ and $\mathbf{Q}$ are fixed:

$$f(p_{ij}) = \|(\mathbf{r}_i^\intercal - \sum_{\tilde{k} \neq j} p_{i\tilde{k}} \mathbf{q}_{\tilde{k}}^\intercal) - p_{ij}\mathbf{q}_j^\intercal\|_2^2 + \text{const}, \quad (7a)$$

$$f(q_{ij}) = \|(\mathbf{r}_j - \sum_{\tilde{k} \neq i} \mathbf{p}_{\tilde{k}} q_{\tilde{k}j}) - \mathbf{p}_i q_{ij}\|_2^2 + \text{const}, \quad (7b)$$

where $\mathbf{r}_i^\intercal$ and $\mathbf{r}_j$ denote the i^{th} row and the j^{th} column of $\mathbf{R}$, respectively. Kim, He and Park, [13] discuss about NMF using BCD method.

B. Bounded Matrix Low Rank Approximation

The building blocks of BMA are column vectors $\mathbf{p}_x$ and row vectors $\mathbf{q}_x^\intercal$ of the matrix $\mathbf{P}$ and $\mathbf{Q}$ respectively. In this section, we discuss the idea behind finding these vectors $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$ such that all the elements in $\mathbf{T} + \mathbf{p}_x\mathbf{q}_x^\intercal \in [r_{min}, r_{max}]$ and the error $\|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{PQ})\|_F^2$ is reduced. Here, $\mathbf{T} = \sum_{j=1, j\neq x}^{k} \mathbf{p}_j\mathbf{q}_j^\intercal$.

The Problem (1) can be equivalently represented with a set of rank-one matrices $\mathbf{p}_x\mathbf{q}_x^\intercal$ as

$$\begin{aligned} \min_{\mathbf{p}_x, \mathbf{q}_x} \quad & \|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{T} - \mathbf{p}_x\mathbf{q}_x^\intercal)\|_F^2 \\ & \forall x = [1, k] \\ \text{subject to} \quad & \\ & \mathbf{T} + \mathbf{p}_x\mathbf{q}_x^\intercal \leq r_{max} \\ & \mathbf{T} + \mathbf{p}_x\mathbf{q}_x^\intercal \geq r_{min} \end{aligned} \quad (8)$$

Thus, we take turns in solving $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$. That is, assume we know $\mathbf{p}_x$ and find $\mathbf{q}_x^\intercal$ and vice versa. Here afterwards, in the entire section we assume fixing column $\mathbf{p}_x$ and finding row $\mathbf{q}_x^\intercal$. Without loss of generality, all the discussions pertaining to finding $\mathbf{q}_x^\intercal$ fixing $\mathbf{p}_x$ hold good for the other scenario of finding $\mathbf{p}_x$ fixing $\mathbf{q}_x^\intercal$.

There are different orders of updates of vector blocks when solving the Problem (8). For example,

$$\mathbf{p}_1 \to \mathbf{q}_1^\intercal \to \cdots \to \mathbf{p}_k \to \mathbf{q}_k^\intercal \quad (9)$$

and

$$\mathbf{p}_1 \to \cdots \to \mathbf{p}_k \to \mathbf{q}_1^\intercal \to \cdots \to \mathbf{q}_k^\intercal. \quad (10)$$

Kim, He and Park [13] prove that equation (6) satisfies the formulation of BCD method. Equation (6) when extended with the matrix $\mathbf{M}$ becomes equation (8). Here, the matrix $\mathbf{M}$ is like a filter matrix that defines the elements of $(\mathbf{R} - \mathbf{T} - \mathbf{p}_x\mathbf{q}_x^\intercal)$ to be included for the norm computation. Thus, the Problem (8) is similar to Problem (6) and we can solve by applying $2k$ block BCD to update $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$ iteratively, although equation (8) appears not to satisfy the BCD requirements directly. We focus on scalar block case, since by imposing the lower and upper bounds to the lower rank matrix, the problem can be more easily understood.

Also, according to BCD, if the solutions of the elements in a block are independent, they can be computed simultaneously. Here, the elements $q_{xi}, q_{xj} \in \mathbf{q}_x^\intercal, i \neq j$, are independent of each other. Hence, the problem of finding row $\mathbf{q}_x^\intercal$ fixing column $\mathbf{p}_x$ is equivalent to solving the following problem

$$\begin{aligned} & \min_{q_{xi}} \|\mathbf{M}(:, i) \cdot *((\mathbf{R} - \mathbf{T})(:, i) - \mathbf{p}_x q_{xi})\|_F^2 \\ & \forall i = [1, m], \forall x = [1, k] \\ \text{subject to} \quad & \\ & \mathbf{T}(:, i) + \mathbf{p}_x q_{xi} \leq r_{max} \\ & \mathbf{T}(:, i) + \mathbf{p}_x q_{xi} \geq r_{min} \end{aligned} \quad (11)$$

To construct the row vector $\mathbf{q}_x^\intercal$, we use $k(n+m)$ scalar blocks based on the problem formulation (11). Theorem 3 identifies these best elements that construct $\mathbf{q}_x^\intercal$. Given $\mathbf{T}$, $\mathbf{R}$ and assume we are fixing column $\mathbf{p}_x$, we find the row vector $\mathbf{q}_x^\intercal = (q_{x1}, q_{x2}, \cdots, q_{xm})$ for the Problem (11). For this, let us understand the boundary values of q_{xi} by defining two bounding vectors one for each above and below.

Definition 1: The lower bound vector $\mathbf{l} \in \mathbb{R}^n$ and an upper bound vector $\mathbf{u} \in \mathbb{R}^n$ for a given $\mathbf{p}_x$ and $\mathbf{T}$ that bounds the q_{xi} is defined as,

$$l_j = \begin{cases} \dfrac{r_{min} - \mathbf{T}(j,i)}{p_{jx}} & p_{jx} > 0, \forall j \in [1,n] \\ \dfrac{r_{max} - \mathbf{T}(j,i)}{p_{jx}} & p_{jx} < 0, \forall j \in [1,n] \\ -\infty & otherwise \end{cases}$$

and,

$$u_j = \begin{cases} \dfrac{r_{max} - \mathbf{T}(j,i)}{p_{jx}}, & p_{jx} > 0, \forall j \in [1,n] \\ \dfrac{r_{min} - \mathbf{T}(j,i)}{p_{jx}} & p_{jx} < 0, \forall j \in [1,n] \\ \infty & otherwise \end{cases}$$

It is important to observe that the defined $\mathbf{l}$ and $\mathbf{u}$ are for a given $\mathbf{p}_x$ and $\mathbf{T}$ to bound q_{xi}. Alternatively, if we are solving $\mathbf{p}_x$ for a given $\mathbf{T}$ and $\mathbf{q}_x$, the above function correspondingly represent the possible lower and upper bounds for p_{ix}, where $\mathbf{l}, \mathbf{u} \in \mathbb{R}^m$.

Theorem 2: Given $\mathbf{R}$, $\mathbf{T}$, $\mathbf{p}_x$, the q_{xi} is always bounded as $max(\mathbf{l}) \leq q_{xi} \leq min(\mathbf{u})$.

Proof: It is easy to see that if $q_{xi} < max(\mathbf{l})$ or $q_{xi} > min(\mathbf{u})$, then $\mathbf{T}(:,i) + \mathbf{p}_x q_{xi}^\intercal \notin [r_{min}, r_{max}]$. ■

Here, it is imperative to note that if q_{xi}, results in $\mathbf{T}(:,i) + \mathbf{p}_x q_{xi}^\intercal \notin [r_{min}, r_{max}]$, this implies that q_{xi} is either less than the $max(\mathbf{l})$ or greater than the $min(\mathbf{u})$. It cannot be any other inequality.

Given the boundary values of q_{xi}, Theorem 3 defines the solution to the Problem (11).

Theorem 3: Given $\mathbf{T}, \mathbf{R}, \mathbf{p}_x, \mathbf{l}$ and $\mathbf{u}$, the unique solution q_{xi} to least squares Problem (11) is given as

$$q_{xi} = \begin{cases} max(\mathbf{l}) : \text{if } q_{xi} < max(\mathbf{l}) \\ min(\mathbf{u}) : \text{if } q_{xi} > min(\mathbf{u}) \\ ([\mathbf{M}(:,i) \cdot *(\mathbf{R} - \mathbf{T})(:,i)]^\intercal \mathbf{p}_x)/(\|\mathbf{M}(:,i) \cdot * \mathbf{p}_x\|_F^2) \\ \qquad\qquad\qquad\qquad : otherwise \end{cases}$$

Proof:

$\underline{\text{Out of Boundary} : q_{xi} < max(\mathbf{l}) \ \& \ q_{xi} > min(\mathbf{u})}$

Under this circumstance, the best value for q_{xi} is either $max(\mathbf{l})$ or $min(\mathbf{u})$. We can prove this by contradiction. Let us assume there exists a $\tilde{q_{xi}} = max(\mathbf{l}) + \delta; \delta > 0$ that is optimal to the Problem (11) for $q_{xi} < max(\mathbf{l})$. However, for $q_{xi} = max(\mathbf{l}) < \tilde{q_{xi}}$ is still a feasible solution for the Problem (11). Also there does not exist a feasible solution that is less than $max(\mathbf{l})$, because the Problem (11) is quadratic in q_{xi}. Hence for $q_{xi} < max(\mathbf{l})$, the optimal value for the Problem (11) is $max(\mathbf{l})$. In similar direction we can show that optimal value of q_{xi} is $min(\mathbf{u})$ for $q_{xi} > min(\mathbf{u})$.

$\underline{\text{Within Boundary} : max(\mathbf{l}) \leq q_{xi} \leq min(\mathbf{u})}$

The derivative of the objective function in the optimization Problem (11) results as

$2\|\mathbf{M}(:,i). * \mathbf{p}_x\|_F^2 q_{xi} - 2[\mathbf{M}(:,i). * (\mathbf{R} - \mathbf{T})(:,i)]^\intercal \mathbf{p}_x$. ■

V. Implementations

Now we have necessary tools to construct the algorithm. The BMA has three major functions. (1) Initialization, (2) Stopping Criteria and (3) Find the low rank factors $\mathbf{P}$ and $\mathbf{Q}$. The initialization and the stopping criteria are explained in further detail in the later sections. For the time being, let us understand that we need two initial matrices $\mathbf{P}$ and $\mathbf{Q}$, such that $\mathbf{PQ} \in [r_{min}, r_{max}]$. And we need a stopping criteria for terminating the algorithm, when the constructed matrices $\mathbf{P}$ and $\mathbf{Q}$ provide a good representation of the given matrix $\mathbf{R}$.

In the case of BMA algorithm, since multiple elements can be updated independently, we reorganize the scalar block BCD into $2k$ vector blocks. The BMA algorithm is presented to the readers as Algorithm 1.

Algorithm 1 works very well and yields low rank factorized matrices $\mathbf{P}$ and $\mathbf{Q}$ for a given matrix $\mathbf{R}$ such that $\mathbf{PQ} \in [r_{min}, r_{max}]$. However, when applied for very large scale matrices, such as recommender systems, it can only be run on machines with a large amount of memory. We address scaling the algorithm in multi core systems and machines with low memory in the next section.

A. Scalable Bounded Matrix Low Rank Approximation

In this section, we address the issue of scaling the algorithm for large matrices with missing elements. The two important aspects of making the algorithm run for large matrices are running time and memory. We discuss the parallel implementation of the algorithm, which we refer to as *Parallel Bounded Matrix Low Rank Approximation*. Subsequently, we also discuss a method called *Block Bounded Matrix Low Rank Approximation*, which will outline the details of executing the algorithm for large matrices in low memory systems. Let us start this section by discussing *Parallel Bounded Matrix Low Rank Approximation*.

1) Parallel Bounded Matrix Low Rank Approximation: In the case of BCD method, the solutions of sub-problems that depend on each other have to be computed sequentially to make use of the most recent values. However, if solutions for some blocks are independent from each other, simultaneous computation of them would be possible. We can observe that, according to Theorem 3, every element $q_{xi}, q_{xj} \in \mathbf{q}_x^\intercal, i \neq j$ is independent of each other. We are leveraging this characteristic to parallelize the **for** loop in the Algorithm 1. Now-a-days all the commercial processors have multiple cores. Hence, we can parallelize find q_{xi} across multiple cores. We are not presenting the algorithm, as it is trivial to change the **for** in step 12 and step 20 of Algorithm 1 to **parallel for**.

```
input  : Matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, $r_{min}, r_{max} > 1$, reduced
         rank $k$
output: Matrix $\mathbf{P} \in \mathbb{R}^{n \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times m}$
// Rand initialization of P, Q.
1  Initialize $\mathbf{P}$, $\mathbf{Q}$ as a nonnegative random matrix ;
   // modify random $\mathbf{PQ}$ such that
   // $\mathbf{PQ} \in [r_{min}, r_{max}]$
   // maxelement of $\mathbf{PQ}$ without first
   // column of $\mathbf{P}$ and first row of $\mathbf{Q}$
2  $maxElement = max(\mathbf{P}(:, 2 : end) * \mathbf{Q}(2 : end, :))$;
3  $\alpha \leftarrow \sqrt{\frac{r_{max} - 1}{maxElement}}$;
4  $\mathbf{P} \leftarrow \alpha \cdot \mathbf{P}$;
5  $\mathbf{Q} \leftarrow \alpha \cdot \mathbf{Q}$;
6  $\mathbf{P}(:, 1) \leftarrow 1$;
7  $\mathbf{Q}(1, :) \leftarrow 1$;
8  $\mathbf{M} \leftarrow$ ComputeRatedBinaryMatrix($\underline{\mathbf{R}}$);
9  while stopping criteria not met do
10     for $x \leftarrow 1$ to $k$ do
11         $\mathbf{T} \leftarrow \sum_{j=1, j \neq x}^{k} \mathbf{p}_j \mathbf{q}_j^\intercal$;
12         for $i \leftarrow 1$ to $m$ do
               // Find vector $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ as in
                  Definition 1
13             $\mathbf{l} \leftarrow$ LowerBounds($r_{min}, r_{max}, \mathbf{T}, i, \mathbf{p}_x$);
14             $\mathbf{u} \leftarrow$
               UpperBounds($r_{min}, r_{max}, \mathbf{T}, i, \mathbf{p}_x$);
               // Find vector $\mathbf{q}_x^\intercal$ fixing $\mathbf{p}_x$ as
                  on Theorem 3
15             $q_{xi} \leftarrow$ FindElement($\mathbf{p_x}, \mathbf{M}, \mathbf{R}, \mathbf{T}, i, x$);
16             if $q_{xi} < max(\mathbf{l})$ then
17                 $q_{xi} \leftarrow max(\mathbf{l})$;
18             else if $q_{xi} > min(\mathbf{u})$ then
19                 $q_{xi} \leftarrow min(\mathbf{u})$;
20         for $i \leftarrow 1$ to $n$ do
               // Find vector $\mathbf{l}, \mathbf{u} \in \mathbb{R}^m$ as in
                  Definition 1
21             $\mathbf{l} \leftarrow$ LowerBounds($r_{min}, r_{max}, \mathbf{T}, i, \mathbf{q}_x^\intercal$);
22             $\mathbf{u} \leftarrow$
               UpperBounds($r_{min}, r_{max}, \mathbf{T}, i, \mathbf{q}_x^\intercal$);
               // Find vector $\mathbf{p}_x$ fixing $\mathbf{q}_x^\intercal$ as
                  on Theorem 3
23             $p_{ix} \leftarrow$
               FindElement($\mathbf{q}_x^\intercal, \mathbf{M}^\intercal, \mathbf{R}^\intercal, \mathbf{T}^\intercal, i, x$);
24             if $p_{ix} < max(\mathbf{l})$ then
25                 $p_{ix} \leftarrow max(\mathbf{l})$;
26             else if $p_{ix} > min(\mathbf{u})$ then
27                 $p_{ix} \leftarrow min(\mathbf{u})$;
```

Algorithm 1: Bounded Matrix Low Rank Approximation (BMA)

It is obvious to see that the $\mathbf{T}$ at Step 11 on Algorithm 1 requires the largest amount of memory. Also the function $FindElement$ in step 15 takes a sizeable amount of memory. Hence it is not possible to run the algorithm for large matrices – with rows and columns in the scale of 100,000's, in machines with low memory. Hence we propose the next algorithm Block BMA.

2) Block Bounded Matrix Low Rank Approximation: To help better understanding of this section, let us define β – a data structure in memory factor. That is, maintaining a floating scalar as single, double, sparse matrix with one element or full matrix with one element takes different amount of memory. This is because of the datastructure that is used to represent the number in the memory. Typically, in Matlab, the data structure in memory factor β for full matrix is around 10. Similarly, in Java, the β factor for maintaining a number in an ArrayList is around 8. Let, $memsize(v)$ be the function that returns the approximate memory size of a variable v. Generally, $memsize(v)$ = number of elements in v * size of each element * β. Consider an example of maintaining 1000 floating point numbers on an ArrayList of a Java program. The approximate memory would be 1000*4*8 = 32000 bytes $\approx$ 32MB against 4MB in actual because of the factor β=8.

As discussed earlier, for most of the real world large datasets such as Netflix, Yahoo music, online dating, book crossing, etc., it is impossible to keep the entire matrix $\mathbf{T}$ in memory. Also notice that, according to Theorem 3 and Definition 1, we need only the i-th column of $\mathbf{T}$ to compute q_{xi}. The block size of $q_{x\mathbf{i}}$ to find in one core of the machine is dependent on the size of $\mathbf{T}$ and $FindElements$ that fits in the memory.

On the one hand, partition $\mathbf{q}_x$ to fit the maximum possible $\mathbf{T}$ and $FindElements$ in the entire memory of the system. On the other hand, create very small partitions such that, we can give every core some amount of work so that we don't under utilize the processing capacity of the system. The disadvantage of the former, is that only one core is used. However, in the latter case, there is a significant communication overhead. Figure 1 gives the pictorial view of the Block Bounded Matrix Low Rank Approximation.

We determined the number of blocks = memsize(full($\mathbf{R}$)+other variables of FindElement)/(system memory * number of d cores). The full($\mathbf{R}$) is non-sparse representation and $d \leq$ number of cores available in the system. Typically for most of the datasets, we achieved minimum running time when we used 4 cores and number of blocks to be 16. That is, we find 1/16-*th* of $\mathbf{q}_x^\intercal$ concurrently on 4 cores.

For the convenience of the readers, we have presented the Block BMA as Algorithm 2. We describe the algorithm only to find the partial vector of $\mathbf{q}_x^\intercal$ given $\mathbf{p}_x$. To find more than one element, Algorithm 1 is modified such that the vectors $\mathbf{l}, \mathbf{u}, \mathbf{p}_x$ are matrices $\mathbf{L}, \mathbf{U}, \mathbf{P}_{blk}$ respectively, in Algorithm

2. The described Algorithm 2 replaces the steps 11 – 19 in Algorithm 1. The initialization and the stopping criteria for Algorithm 2 is similar to those of Algorithm 1. We also include the necessary steps to handle numerical errors as part of Algorithm 2.

```
input  : Matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, set of indices $\mathbf{i}$, current $\mathbf{p}_x$,
         $x$, current $\mathbf{q}'_x$, $r_{min}, r_{max}$
output: Partial vector $\mathbf{q}_x$ of requested indices $\mathbf{i}$
   // ratings of input indices $\mathbf{i}$
1  $\mathbf{R}_{blk} \leftarrow \mathbf{R}(:, \mathbf{i})$ ;
2  $\mathbf{M}_{blk} \leftarrow$ ComputeRatedBinaryMatrix($\underline{\mathbf{R}_{blk}}$) ;
3  $\mathbf{P}_{blk} \leftarrow$ Replicate($\underline{\mathbf{p}_x, size(\mathbf{i})}$) ;
   // save $\mathbf{q}_x$ of input indices $\mathbf{i}$ for
      numerical errors
4  $\mathbf{q}'_{blk} \leftarrow \mathbf{q}_x(\mathbf{i})$ ;
   // $\mathbf{T}_{blk} \in n \times size(\mathbf{i})$ of input indices $\mathbf{i}$
5  $\mathbf{T}_{blk} \leftarrow \sum_{j=1, j \neq x}^{k} \mathbf{p}_j \mathbf{q}'^{\intercal}_{blk}$;
   // Find matrix $\mathbf{L}, \mathbf{U} \in \mathbb{R}^{n \times size(\mathbf{i})}$ as in
      Definition 1
6  $\mathbf{L} \leftarrow$ LowerBounds($\underline{r_{min}, r_{max}, \mathbf{T}, \mathbf{i}, \mathbf{p}_x}$) ;
7  $\mathbf{U} \leftarrow$ UpperBounds($\underline{r_{min}, r_{max}, \mathbf{T}, \mathbf{i}, \mathbf{p}_x}$) ;
   // Find vector $\mathbf{q}_{blk}$ fixing $\mathbf{p}_x$ as in
      Theorem 3
8  $\mathbf{q}_{blk} = ([\mathbf{M}_{blk} .* (\mathbf{R}_{blk} - \mathbf{T}_{blk})]^{\intercal} \mathbf{P}_{blk}) / (\|\mathbf{M}_{blk} .* \mathbf{P}_{blk}\|_F^2)$
   ;
   // indices of $\mathbf{q}_{blk}$ that are not within
      bounds
9  $\mathbf{idxlb} \leftarrow find(\mathbf{q}_{blk} < max(\mathbf{L}))$ ;
10 $\mathbf{idxub} \leftarrow find(\mathbf{q}_{blk} > min(\mathbf{U}))$ ;
   // case 1 & 2 numerical errors
11 $idxcase1 \leftarrow find([\mathbf{q}'_{blk} \approx max(\mathbf{L})] or [\mathbf{q}'_{blk} \approx min(\mathbf{U})])$
   ;
12 $idxcase2 \leftarrow find([max(\mathbf{L}) \approx min(\mathbf{U})] or [max(\mathbf{L}) >$
   $min(\mathbf{U})])$ ;
13 $idxdontchange \leftarrow idxcase1 \cup idxcase2$;
   // set appropriate values of
      $\mathbf{q}_{blk} \notin [max(\mathbf{L}), min(\mathbf{U})]$
14 $\mathbf{q}_{blk}(\mathbf{idxlb} \setminus idxdontchange) \leftarrow$
   $max(\mathbf{L})(idxlb \setminus idxdontchange)$ ;
15 $\mathbf{q}_{blk}(\mathbf{idxub} \setminus idxdontchange) \leftarrow$
   $min(\mathbf{U})(idxub \setminus idxdontchange)$ ;
16 $\mathbf{q}_{blk}(idxdontchange) \leftarrow \mathbf{q}'_{blk}(idxdontchange)$ ;
```

Algorithm 2: Block BMA

In the next section, we discuss tuning various parameters in the algorithm for improving the accuracy for prediction tasks.

B. Parameter Tuning

The BMA has many types of applications. One important application of the BMA is prediction of the missing elements in the matrix. For example, in the case of recommender systems the missing ratings are provided as ground truth in the form of a test data. The dot product of $\mathbf{P}(u,:)$ and $\mathbf{Q}(:,i)$ gives the missing rating of a (u,i) pair. In such cases, the accuracy of the algorithm is determined by the Root Mean Square Error (RMSE) of the predicted ratings against

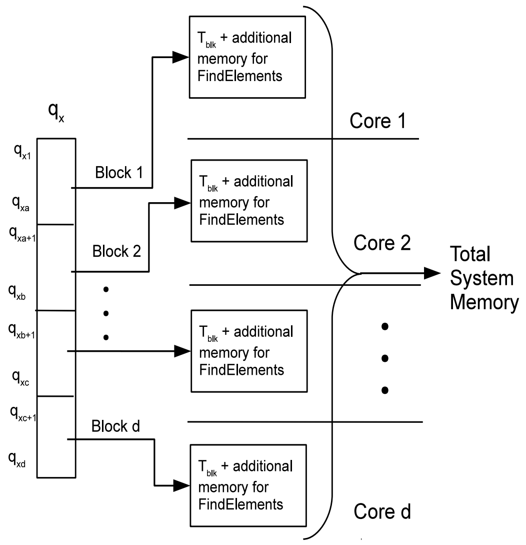

Figure 1: Block Bounded Matrix Low Rank Approximation

the ground truth. It is unimportant how good the algorithm converges for a given rank k.

This section discusses ways to improve the RMSE of the predictions against the missing ratings by tuning the parameters of the BMA algorithm.

1) Initialization: The BMA algorithm can converge to different points depending on the initialization. In Algorithm 1, we explained about random initialization such that $\mathbf{PQ} \in [r_{min}, r_{max}]$. In general, it should give good results.

However, in the case of recommender system, we tuned this initialization for better results. According to Koren [15], one good baseline estimate for a missing rating (u,i) is $\mu + g_u + h_i$, where μ is the average of the known ratings, and g_u and h_i are the bias of user u and item i, respectively. We initialized $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PQ}(u,i) = \mu + g_u + h_i$ as mentioned below

$$\mathbf{P} = \begin{pmatrix} \frac{\mu}{k-2} & \frac{\mu}{k-2} & \cdots & g_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\mu}{k-2} & \frac{\mu}{k-2} & \cdots & g_n & 1 \end{pmatrix}$$

and

$$\mathbf{Q} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \\ h_1 & h_2 & \cdots & h_m \end{pmatrix}$$

That is, let the first $k-2$ columns of $\mathbf{P}$ be $\frac{\mu}{k-2}$, $\mathbf{P}(:, k-1) = \mathbf{g}$ and $\mathbf{P}(:,k) = 1$. Let all the $k-1$ rows of $\mathbf{Q}$ be 1's and $\mathbf{Q}(k,:) = \mathbf{h}^{\intercal}$. We call this as baseline initialization and expected to work better than random initializations for certain type of and vice versa.

2) Reduced Rank k: In the case of regular low rank approximation with all known elements, the higher the k; the closer the low rank approximation to the input matrix [13]. However, in the case of predicting with the low rank factors, a good k depends on the nature of the dataset. Even though, for a higher k, the low rank approximation is closer to the known rating of the input $\mathbf{R}$, the RMSE on the test data may be poor. In the Table III, we can observe the behaviour of RMSE on the test data against k. Most of the time, a good k is determined by trial and error for the prediction problem.

3) Stopping Criteria $\mathfrak{C}$: The stopping criteria defines the goodness of the low rank approximation for the given matrix and the task for which the low rank factors are used. The two common stopping criteria are – (1) For a given rank k, the product of the low rank factors should be closer to the known ratings of the matrix and (2) The low rank factors should perform the prediction task on a smaller validation set which has the same distribution of the test set. The former is common when all the elements of $\mathbf{R}$ are known and the latter for recommender system.

Let us formally define the stopping criteria $\mathfrak{C}1$ for the former case as $\sqrt{\dfrac{\|\mathbf{M}\cdot*(\mathbf{R}-\mathbf{PQ})\|_F^2}{numRatings\ in\ \mathbf{R}}}$ did not change in the successive iterations at a given floating point precision like 1e-5. Since the function is monotonically decreasing, for every iteration, we find $\mathbf{P}$ and $\mathbf{Q}$ that improve the solution to the Problem (1). Otherwise, we terminate.

The stopping criteria $\mathfrak{C}2$ for the latter case is the increase of $\sqrt{\dfrac{\|\mathbf{M}\cdot*(\mathbf{V}-\mathbf{PQ})\|_F^2}{numRatings\ in\ \mathbf{V}}}$, for some validation matrix $\mathbf{V}$ which has the same distribution of the test matrix, between successive iterations. Here, $\mathbf{M}$ is for the validation matrix $\mathbf{V}$. This stopping criteria has diminishing effect as the number of iterations increase. Hence, we also check whether $\sqrt{\dfrac{\|\mathbf{M}\cdot*(\mathbf{V}-\mathbf{PQ})\|_F^2}{numRatings\ in\ \mathbf{V}}}$ did not change in the successive iterations at a given floating point precision like 1e-5.

We can show that, for the above stopping criterion $\mathfrak{C}1$ and $\mathfrak{C}2$, the Algorithm 1 terminates for any input matrix $\mathbf{R}$.

Theorem 4: Algorithm 1 terminates for both the stopping criterion $\mathfrak{C}1$ and $\mathfrak{C}2$.

Proof: Case 1 : Termination with $\mathfrak{C}1$: It is trivial that the $\|\mathbf{R}-\mathbf{T}\|_F$ is monotonically decreasing for every $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$. Hence at the end of completely finding all the vectors of $\mathbf{P}$ and $\mathbf{Q}$, $\|\mathbf{M}.*(\mathbf{R}-\mathbf{PQ})\|_F$ would have decreased. In other words, we find only those $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$ that reduce the RMSE between the known ratings in the rating matrix and the corresponding factors of $\mathbf{P}$ and $\mathbf{Q}$. Alternatively, if the algorithm could not find $\mathbf{P}$ and $\mathbf{Q}$ due to convergence, the $\|\mathbf{M}.*(\mathbf{R}-\mathbf{PQ})\|_F$ does not change. Hence, the algorithm terminates.

Dataset	Rows	Columns	Ratings (millions)	Density	Ratings Range
Jester	73421	100	4.1	0.5584	[-10,10]
Movielens	71567	10681	10	0.0131	[1,5]
Dating	135359	168791	17.3	0.0007	[1,10]
Book crossing	278858	271379	1.1	0.00001	[1,10]

Table II: Datasets for experimentation

Case 2 : Termination with $\mathfrak{C}2$: It is trivial because at the end of the iteration, we terminate if the RMSE on the validation set has either increased or decreased marginally. ■

VI. Experimentation

In the experimentation, we take an important application of the BMA – recommender system. The entire experimentation was conducted with Algorithm 2 in various systems with memory as lows as 16GB.

One of the major challenges during experimentation is numerical errors. The numerical errors might result in $\mathbf{T}+\mathbf{p}_x\mathbf{q}_x^\intercal \notin [r_{min}, r_{max}]$. The two fundamental questions to solve the numerical errors are – (1) How to identify the occurrence of a numerical error? and (2) What is the best possible value to choose in the case of a numerical error?

Let us start with addressing the former question of potential numerical errors that arise in the BMA Algorithm 1. It is important to understand that if we are well within bounds, i.e., if $max(\mathbf{l}) < q_{xi} < min(\mathbf{u})$, we are not essentially impacted by the numerical errors. It is critical only when q_{xi} is out of boundary, that is, $q_{xi} < max(\mathbf{l})$ or $q_{xi} > min(\mathbf{u})$ and approximately closer to the boundary discussed as in (*Case A* and *Case B*). For discussion let us assume we are improving the old value of q'_{xi} to q_{xi} such that we minimize the error $\|\mathbf{M}\cdot*(\mathbf{R}-\mathbf{T}-\mathbf{p}_x q_x^\intercal)\|_F^2$.

Case A: $q'_{xi} \approx max(\mathbf{l})$ or $q'_{xi} \approx min(\mathbf{u})$: This is equivalent to saying q'_{xi} is already optimal for the given $\mathbf{p}_x$ and $\mathbf{T}$ and there is no further improvement possible. Under this scenario, if $q'_{xi} \approx q_{xi}$ and it is better to retain q'_{xi} irrespective of the new q_{xi} found.

Case B: $max(\mathbf{l}) \approx min(\mathbf{u})$ or $max(\mathbf{l}) > min(\mathbf{u})$: According to Theorem 2, we know that $max(\mathbf{l}) < min(\mathbf{u})$. Hence if $max(\mathbf{l}) > min(\mathbf{u})$, it is only because of the numerical errors.

In all the above cases during numerical errors, we are better off retaining the old value q'_{xi} against the new value q_{xi}. We have explained the Algorithm 2 – Block BMA considering the numerical errors.

We experimented this Algorithm 2 among varied bounds in very large matrix sizes taken from the real world datasets. The datasets used for our experiments include the movielens 10 million [1], jester [8], book crossing [27] and online dating dataset [3]. The characteristics of the dataset are presented in Table II.

We have chosen Root Mean Square Error (RMSE) – a defacto metric for recommender systems. We compare the RMSE of BMA with baseline initialization (BMA–Baseline) and BMA with random initialization (BMA–Random) against the other algorithms on all the datasets. The algorithms used for comparison are ALSWR (alternating least squares with regularization) [26], SGD [7], SVD++ [15] and Bias-SVD [15] and its implementation in Graphlab(http://graphlab.org/) [20] software package. We implemented our algorithm in Matlab and used parallel computing toolbox for parallelizing across multiple cores.

For parameter tuning, we varied the number of reduced rank k and tried different initial matrices for our algorithm to compare against all the algorithms. Many algorithms do not support the stopping criteria $\mathfrak{C}1$ and hence, to be consistent, we used the stopping criteria $\mathfrak{C}2$.

For every k, every dataset was randomly partitioned into 85% training, 5% validation and 10% test data. We run all the algorithms on these partitions and computed their RMSE scores. We repeated each experiment 5 times and reported their RMSE scores in the Table III, where each resulting value is the average of the RMSE scores on randomly chosen test set on 5 runs. The Table III summarizes the RMSE comparison of all the algorithms.

The Algorithm 1 consistently outperformed existing state of the art algorithms. One of the main reason for the consistent performance is the absence of hyper parameters. In the case of machine learning algorithms, there are many parameters that need to be tuned for performance. Even though the algorithms perform the best when provided with the right parameters, identifying these parameters are formidable challenge – usually by trial and error methods. For example, in Table III, we can observe that the Bias-SVD an algorithm without hyper parameters performed better than its extension SVD++ with default parameters in many cases. The BMA algorithm without hyper parameters performed well in real world datasets, albeit a BMA with hyper parameters and right parametric values would have performed better.

Recently, there has been a surge in interest to understand the temporal impact on the ratings. Time-svd++ [16] is one such algorithm that leverages the time of rating to improve prediction accuracy. Also, the most celebrated dataset in the recommender system community is Netflix dataset, because of the prize money and first massive dataset for recommender system that was publicly made available. In Netflix dataset there were 17770 users who rated 480189 movies in a scale of [1,5]. There were totally 100,480,507 ratings in training set and 1,408,342 ratings in validation set. All the algorithms listed above were invented to address Netflix challenge. Even though the book crossing dataset [27] is bigger than the Netflix, we felt the paper is not complete without experimenting on Netflix and comparing against time-SVD++. However, the major challenge is that Netflix dataset has been withdrawn from the internet and its test data is no more available. Hence, we extracted a small sample of 5% from the training data as validation set and tested the algorithm against the validation set that was supplied as part of the training package. We performed this experiment and results are presented in Table IV. For better comparison we also present the original Netflix test scores for SVD++ and time-SVD++ algorithms from [16]. These are labeled as *SVD++-Test* and *time-SVD++-Test* respectively. Our BMA algorithm out performed all the algorithm in Netflix dataset when tested on the validation set supplied as part of the Netflix training package.

Algorithm	$k = 10$	$k = 20$	$k = 50$	$k = 100$
BMA Baseline	0.9521	0.9533	0.9405	0.9287
BMA Random	0.9883	0.9569	0.9405	0.8777
ALSWR	1.5663	1.5663	1.5664	1.5663
SVD++	1.6319	1.5453	1.5235	1.5135
SGD	1.2997	1.2997	1.2997	1.2997
Bias-SVD	1.3920	1.3882	1.3662	1.3354
time-svd++	1.1800	1.1829	1.1884	1.1868
SVD++-Test	0.9131	0.9032	0.8952	0.8924
time-SVD++-Test	0.8971	0.8891	0.8824	0.8805

Table IV: RMSE Comparison of BMA with other algorithms on Netflix

VII. Conclusion

In this paper, we presented a new low rank approximation called Bounded Matrix Low Rank Approximation (BMA) which imposed a lower and upper bound on every element of a lower rank matrix that best approximates a given matrix with missing elements. We also presented substantial experimental results on real world datasets illustrating that our proposed method outperformed the state of the art algorithms in recommender systems. We would like to extend BMA to tensors, such that it can use time information of the ratings during factorization. During our experimentation, we observed linear scale up for Algorithm 2 in Matlab. However, the other algorithms from Graphlab are implemented in C/C++ and take lesser clock time. A C/C++ implementation of Algorithm 2 would be a good starting point to compare the running time performance against the other state of the art algorithms. Also, we want to experiment BMA in datasets of different nature other than those from recommender systems.

Acknowledgement

This work is supported in part by the National Science Foundation grants CCF-0732318 and CCF-0808863. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundations.

Dataset	k	BMA Baseline	BMA Random	ALSWR	SVD++	SGD	Bias-SVD
Jester	10	4.3320	4.6289	5.6423	5.5371	5.7170	5.8261
Jester	20	4.3664	4.7339	5.6579	5.5466	5.6752	5.7862
Jester	50	4.5046	4.7180	5.6713	5.5437	5.6689	5.7956
Movielens10M	10	0.8531	0.8974	1.5166	1.4248	1.2386	1.2329
Movielens10M	20	0.8526	0.8931	1.5158	1.4196	1.2371	1.2317
Movielens10M	50	0.8553	0.8932	1.5162	1.4204	1.2381	1.2324
Dating	10	1.9309	2.1625	3.8581	4.1902	3.9082	3.9052
Dating	20	1.9337	2.1617	3.8643	4.1868	3.9144	3.9115
Dating	50	1.9434	2.1642	3.8606	4.1764	3.9123	3.9096
Book Crossing	10	1.9355	2.8137	4.7131	4.7315	5.1772	3.9466
Book Crossing	20	1.9315	2.4652	4.7212	4.6762	5.1719	3.9645
Book Crossing	50	1.9405	2.1269	4.7168	4.6918	5.1785	3.9492

Table III: RMSE Comparison of Algorithms on Real World Datasets

REFERENCES

[1] Movielens dataset. http://movielens.umn.edu. [Online; accessed 6-June-2012].

[2] D. P. Bertsekas. Nonlinear Programming. Athena Scientific, Belmont, MA, 1999.

[3] L. Brozovsky and V. Petricek. Recommender system for online dating service. In Proceedings of Conference Znalosti 2007, Ostrava, 2007. VSB.

[4] A. Cichocki and A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E92-A:708–721, 2009.

[5] A. Cichocki, R. Zdunek, and S. Amari. Hierarchical als algorithms for nonnegative matrix and 3d tensor factorization. LNCS, 4666:169–176, 2007.

[6] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41:391–407, 1990.

[7] S. Funk. Stochastic gradient descent. http://sifter.org/~simon/journal/20061211.html, 2006. [Online; accessed 6-June-2012].

[8] K. Goldberg. Jester collaborative filtering dataset. http://goldberg.berkeley.edu/jester-data/, 2003. [Online; accessed 6-June-2012].

[9] G. H. Golub and C. F. Van Loan. Matrix Computations. The Johns Hopkins University Press, 3rd edition, 1996.

[10] N.-D. Ho, P. V. Dooren, and V. D. Blondel. Descent methods for nonnegative matrix factorization. CoRR, abs/0801.3199, 2008.

[11] H. Kim and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. Bioinformatics, 23(12):1495–1502, 2007.

[12] H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. SIAM Journal on Matrix Analysis and Applications, 30(2):713–730, 2008.

[13] J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. Under Review, 2012.

[14] J. Kim and H. Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. SIAM J. Scientific Computing, 33(6):3261–3281, 2011.

[15] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In KDD, pages 426–434, 2008.

[16] Y. Koren. Collaborative filtering with temporal dynamics. In KDD, page 447, 2009.

[17] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. Computer, 42(8):30–37, Aug. 2009.

[18] D. Kuang, H. Park, and C. H. Q. Ding. Symmetric nonnegative matrix factorization for graph clustering. In SDM, pages 106–117, 2012.

[19] C. J. Lin. Projected Gradient Methods for Nonnegative Matrix Factorization. Neural Comput., 19(10):2756–2779, Oct. 2007.

[20] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. In UAI, 2010.

[21] L. W. Mackey, D. Weiss, and M. I. Jordan. Mixed membership matrix factorization. In ICML, pages 711–718, 2010.

[22] I. Markovsky. Algorithms and literate programs for weighted low-rank approximation with missing data. In J. Levesley, A. Iske, and E. Georgoulis, editors, Approximation Algorithms for Complex Systems, number 18296, pages 255–273. Springer-Verlag, 2011. Chapter: 12.

[23] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In KDD, pages 39–42, 2007.

[24] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In ICML, pages 880–887, 2008.

[25] L. Xiong, X. Chen, T.-K. Huang, J. G. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In SDM, pages 211–222, 2010.

[26] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale Parallel Collaborative Filtering for the Netflix Prize. Algorithmic Aspects in Information and Management, 5034:337–348, 2008.

[27] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In WWW, pages 22–32, 2005.